고객을 세그먼테이션하자 [프로젝트]

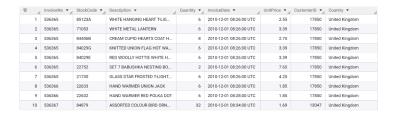
11-2. 데이터 불러오기

데이터 살펴보기

• 테이블에 있는 10개의 행만 출력하기

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.data`
LIMIT 10;
```

[결과]



• 전체 데이터는 몇 행으로 구성되어 있는지 확인하기

```
SELECT COUNT(*)
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]



데이터 수 세기

• COUNT 함수를 사용해서, 각 컬럼별 데이터 포인트의 수를 세어 보기

```
SELECT COUNT(InvoiceNo) InvoiceNo, COUNT(StockCode) StockCode, COUNT(Description) Description, COUNT(Quantity) Quantity, COUNT(InvoiceDate) InvoiceDate, COUNT(UnitPrice) UnitPrice, COUNT(CustomerID) CustomerID, COUNT(Country) Country FROM `proven-hash-466600-d5.modulabs_project.data`;
```

[결과]



11-4. 데이터 전처리 방법(1): 결측치 제거

컬럼 별 누락된 값의 비율 계산

- 각 컬럼 별 누락된 값의 비율을 계산
 - 。 각 컬럼에 대해서 누락 값을 계산한 후, 계산된 누락 값을 UNION ALL을 통해 합치기

SELECT column_name, ROUND((total - column_value) / total * 100, 2)

FROM

SELECT 'InvoiceNo' AS column_name, COUNT(InvoiceNo) AS column_value, COUNT(*) AS total FROM `proven-hash-466600-UNION ALL

SELECT 'StockCode' AS column_name, COUNT(StockCode) AS column_value, COUNT(*) AS total FROM 'proven-hash-46660 UNION ALL

SELECT 'Description' AS column_name, COUNT(Description) AS column_value, COUNT(*) AS total FROM `proven-hash-4666 UNION ALL

SELECT 'Quantity' AS column_name, COUNT(Quantity) AS column_value, COUNT(*) AS total FROM `proven-hash-466600-d5 LINION ALL

SELECT 'InvoiceDate' AS column_name, COUNT(InvoiceDate) AS column_value, COUNT(*) AS total FROM `proven-hash-4666 UNION ALL

SELECT 'UnitPrice' AS column_name, COUNT(UnitPrice) AS column_value, COUNT(*) AS total FROM `proven-hash-466600-d UNION ALL

SELECT 'CustomerID' AS column_name, COUNT(CustomerID) AS column_value, COUNT(*) AS total FROM `proven-hash-4666 UNION ALL

SELECT 'Country' AS column_name, COUNT(Country) AS column_value, COUNT(*) AS total FROM `proven-hash-466600-d5.i

[결과]

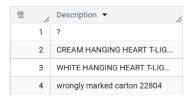
행 //	column_name ▼	f0_ ▼
1	CustomerID	24.93
2	Country	0.0
3	UnitPrice	0.0
4	Description	0.27
5	InvoiceNo	0.0
6	InvoiceDate	0.0
7	StockCode	0.0
8	Quantity	0.0

결측치 처리 전략

• StockCode = '85123A' 의 Description 을 추출하는 쿼리문을 작성하기

SELECT DISTINCT Description
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE StockCode = '85123A'
ORDER BY Description;

[결과]



결측치 처리

• DELETE 구문을 사용하며, WHERE 절을 통해 데이터를 제거할 조건을 제시

DELETE

FROM `proven-hash-466600-d5.modulabs_project.data` WHERE CustomerID IS NULL OR Description IS NULL;

-- 검증
SELECT COUNTIF(InvoiceNo IS NULL) InvoiceNo, COUNTIF(StockCode IS NULL) StockCode,
COUNTIF(Description IS NULL) Description, COUNTIF(Quantity IS NULL) Quantity,
COUNTIF(InvoiceDate IS NULL) InvoiceDate, COUNTIF(UnitPrice IS NULL) UnitPrice,
COUNTIF(CustomerID IS NULL) CustomerID, COUNTIF(Country IS NULL) Country
FROM `proven-hash-466600-d5.modulabs_project.data`;

[결과]



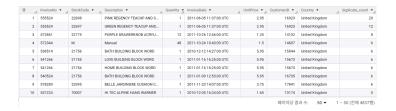
11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

- 중복된 행의 수를 세어보기
 - 8개의 컬럼에 그룹 함수를 적용한 후, COUNT가 1보다 큰 데이터를 세어보기

SELECT
InvoiceNo, StockCode, Description, Quantity, InvoiceDate,
UnitPrice, CustomerID, Country,
COUNT(*) AS duplicate_count
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY
InvoiceNo, StockCode, Description, Quantity, InvoiceDate,
UnitPrice, CustomerID, Country
HAVING COUNT(*) > 1
ORDER BY duplicate_count DESC;

[결과]



중복값 처리

- 중복값을 제거하는 쿼리문 작성하기
 - \circ CREATE OR REPLACE TABLE 구문을 활용하여 모든 컬럼(*)을 DISTINCT 한 데이터로 업데이트

CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.data` AS SELECT DISTINCT * $FROM `proven-hash-466600-d5.modulabs_project.data`;$

[결과]



11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

• 고유(unique)한 InvoiceNo 의 개수를 출력하기

SELECT COUNT(DISTINCT InvoiceNo) AS unique_InvoiceNo FROM `proven-hash-466600-d5.modulabs_project.data`;

[결과]



• 고유한 InvoiceNo 를 앞에서부터 100개를 출력하기

SELECT DISTINCT InvoiceNo AS InvoiceNo100 FROM `proven-hash-466600-d5.modulabs_project.data` LIMIT 100;

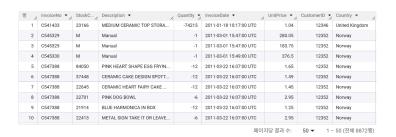
[결과]



• InvoiceNo 가 'C'로 시작하는 행을 필터링 할 수 있는 쿼리문을 작성하기 (100행까지만 출력)

SELECT *
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;

[결과]



• 구매 건 상태가 Canceled 인 데이터의 비율(%) - 소수점 첫번째 자리까지

SELECT

 $ROUND(COUNTIF(InvoiceNo\ LIKE\ 'C\%')*100.0\ /\ COUNT(*),\ 1)\ AS\ canceled_ratio\\ FROM\ `proven-hash-466600-d5.modulabs_project.data`;$

[결과]



StockCode 살펴보기

• 고유한 StockCode 의 개수를 출력하기

SELECT COUNT(DISTINCT StockCode) AS unique_stockcode FROM `proven-hash-466600-d5.modulabs_project.data`;

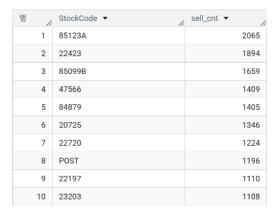
[결과]



- 어떤 제품이 가장 많이 판매되었는지 보기 위하여 StockCode 별 등장 빈도를 출력하기
 - 。 상위 10개의 제품들을 출력하기

SELECT StockCode, COUNT(*) AS sell_cnt FROM `proven-hash-466600-d5.modulabs_project.data` GROUP BY StockCode ORDER BY sell_cnt DESC LIMIT 10;

[결과]



- StockCode 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고
 - 。 **숫자가 0~1개인 값**들에는 어떤 코드들이 들어가 있는지 출력하기

SELECT DISTINCT StockCode, number_count FROM (SELECT StockCode,

LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', ")) AS number_count

```
FROM `proven-hash-466600-d5.modulabs_project.data`)

WHERE number_count <= 1;

WITH UniqueStockCodes AS (
    SELECT DISTINCT StockCode
    FROM `proven-hash-466600-d5.modulabs_project.data`)

SELECT
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count,
    COUNT(*) AS stock_cnt
FROM UniqueStockCodes
GROUP BY number_count
```

[결과]



행 //	StockCode ▼	number_count ▼ //
1	POST	0
2	М	0
3	C2	1
4	D	0
5	BANK CHARGES	0
6	PADS	0
7	DOT	0
8	CRUK	0

- StockCode 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고
 - **숫자가 0~1개인 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트**인지 구하기 (소수점 두 번째 자리까지)

```
WITH Annotated AS (

SELECT *,

LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count

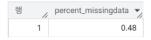
FROM `proven-hash-466600-d5.modulabs_project.data`
)

SELECT

ROUND(COUNTIF(number_count <= 1) * 100.0 / COUNT(*), 2) AS percent_missingdata

FROM Annotated;
```

[결과]



• 제품과 관련되지 않은 거래 기록을 제거하기

```
DELETE
FROM 'proven-hash-466600-d5.modulabs_project.data'
WHERE StockCode IN (
SELECT DISTINCT StockCode
FROM (
SELECT StockCode,
LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
FROM 'proven-hash-466600-d5.modulabs_project.data'
)
```

```
WHERE number_count in (0, 1)
);
```

[결과]

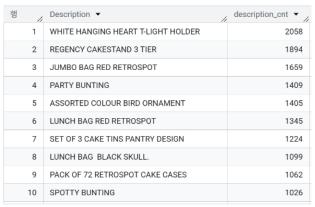
❶ 이 문으로 data의 행 1,915개가 삭제되었습니다.

Description 살펴보기

• 고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기

SELECT Description, COUNT(*) AS description_cnt FROM `proven-hash-466600-d5.modulabs_project.data` GROUP BY Description ORDER BY description_cnt DESC LIMIT 30;

[결과]



페이지당 결과 수: 50 ▼ 1 - 30 (전체 30행)

• 서비스 관련 정보를 포함하는 행들을 제거하기

DELETE
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE description IN ('Next Day Carriage', 'High Resolution Image');

[결과]

● 이 문으로 data의 행 83개가 삭제되었습니다.

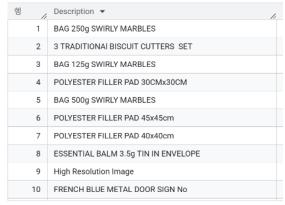
• 대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기

```
-- 대소문자 혼합 확인
SELECT DISTINCT Description
FROM `proven-hash-466600-d5.modulabs_project_.data`
WHERE REGEXP_CONTAINS(Description, r'[a-z]');

-- 대문자로 표준화
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.data` AS
SELECT
* EXCEPT (Description),
UPPER(Description) AS Description
FROM `proven-hash-466600-d5.modulabs_project.data`;
```

-- 확인 SELECT DISTINCT Description FROM `proven-hash-466600-d5.modulabs_project_.data` WHERE REGEXP_CONTAINS(Description, r'[a-z]');

[결과]



페이지당 결과 수: 50 ▼ 1 - 19 (전체 19행)

이 문으로 이름이 data인 테이블이 교체되었습니다.

 표시할 데이터가 없습니다.

UnitPrice 살펴보기

• UnitPrice 의 최솟값, 최댓값, 평균을 구하기

SELECT
MIN(UnitPrice) AS min_price,
MAX(UnitPrice) AS max_price,
ROUND(AVG(UnitPrice), 2) AS avg_price
FROM `proven-hash-466600-d5.modulabs_project.data`;

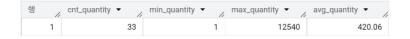
[결과]



• 단가가 0원인 거래의 개수, 구매 수량(Quantity)의 최솟값, 최댓값, 평균 구하기

SELECT
COUNT(*) AS cnt_quantity,
MIN(Quantity) AS min_quantity,
MAX(Quantity) AS max_quantity,
ROUND(AVG(Quantity), 2) AS avg_quantity
FROM 'proven-hash-466600-d5.modulabs_project.data'
WHERE UnitPrice = 0;

[결과]



• UnitPrice = 0 를 제거하고 일관된 데이터셋을 유지하기

```
-- unitprice=0 처리
CREATE OR REPLACE TABLE 'proven-hash-466600-d5.modulabs_project_.data' AS
SELECT *
FROM 'proven-hash-466600-d5.modulabs_project_.data'
WHERE UnitPrice != 0;
-- 확인
SELECT count(unitprice) as cnt_zero_price
FROM 'proven-hash-466600-d5.modulabs_project_.data'
WHERE unitprice = 0;
```

[결과]



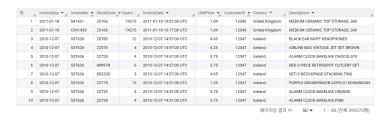
11-7. RFM 스코어

Recency

• InvoiceDate 컬럼을 연월일 자료형으로 변경하기

SELECT DATE(InvoiceDate) AS InvoiceDay, * FROM `proven-hash-466600-d5.modulabs_project.data`;

[결과]



• 가장 최근 구매 일자를 MAX() 함수로 찾아보기

SELECT max(invoiceDate) as most_recent_date
FROM `proven-hash-466600-d5.modulabs_project.data`;

[결과]



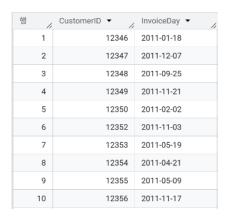
• 유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기

SELECT CustomerID,

MAX(DATE(InvoiceDate)) AS InvoiceDay

FROM `proven-hash-466600-d5.modulabs_project.data` GROUP BY CustomerID;

[결과]



• 가장 최근 일자(most_recent_date)와 유저별 마지막 구매일(InvoiceDay)간의 차이를 계산하기

SELECT CustomerID,
EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (SELECT CustomerID,
MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM `proven-hash-466600-d5.modulabs_project`.data
GROUP BY CustomerID);

[결과]

행 //	CustomerID ▼	recency ▼
1	12349	18
2	12379	81
3	12412	74
4	12490	5
5	12564	261
6	12607	58
7	13017	7
8	13044	291
9	13229	225
10	13255	3

• 최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이고 지금까지의 결과를 user_r 이라는 이름의 테이블로 저장하기

[결과]

● 이 문으로 이름이 user_r인 새 테이블이 생성되었습니다.

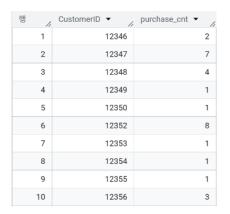
행 //	CustomerID	• //	recency ~	//
1		17001		0
2		15910		0
3		14422		0
4		13069		0
5		18102		0
6		17754		0
7		17428		0
8		14446		0
9		13113		0
10		12985		0

Frequency

• 고객마다 고유한 InvoiceNo의 수를 세어보기

SELECT CustomerID,
COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID;

[결과]



• 각 고객 별로 구매한 아이템의 총 수량 더하기

SELECT CustomerID, SUM(Quantity) AS item_cnt FROM `proven-hash-466600-d5.modulabs_project.data` GROUP BY CustomerID;

[결과]

행 //	CustomerID	· //	item_cnt ▼	11
1		12346		0
2		12347		2458
3		12348		2332
4		12349		630
5		12350		196
6		12352		463
7		12353		20
8		12354		530
9		12355		240
10		12356		1573

• 전체 거래 건수 계산와 구매한 아이템의 총 수량 계산의 결과를 합쳐서 user_rf 라는 이름의 테이블에 저장하기

```
CREATE OR REPLACE TABLE 'proven-hash-466600-d5.modulabs_project.user_rf' AS
WITH purchase_cnt AS (
SELECT
 CustomerID,
 COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID
item_cnt AS (
SELECT
 CustomerID,
 SUM(Quantity) AS item_cnt
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID
SELECT
pc.CustomerID,
pc.purchase_cnt,
ic.item_cnt,
ur.recency
FROM purchase_cnt AS pc
JOIN item_cnt AS ic
ON pc.CustomerID = ic.CustomerID
JOIN `proven-hash-466600-d5.modulabs_project.user_r` AS ur
ON pc.CustomerID = ur.CustomerID;
```

[결과]

a 0	이 문으로 이름이 user_rf인 새 테이블이 생성되었습니다.								
행	CustomerID •	purchase_cnt ▼	/ item_cnt ▼	recency ▼	1				
1	1	2713	1	505	0				
2	1	4569	1	79	1				
3	1	3298	1	96	1				
4	1	5520	1	314	1				
5	1	3436	1	76	1				
6	1	5471	1	256	2				
7	1	5195	1	1404	2				
8	1	4204	1	72	2				
9	1	2650	1	250	3				
10	1	6569	1	93	3				

Monetary

• 고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)

```
SELECT
CustomerID,
ROUND(SUM(Quantity * UnitPrice), 1) AS user_total
FROM `proven-hash-466600-d5.modulabs_projec.data`
WHERE CustomerID IS NOT NULL
GROUP BY CustomerID;
```

[결과]

행 //	CustomerID	· //	user_total ▼
1		12346	0.0
2		12347	4310.0
3		12348	1437.2
4		12349	1457.6
5		12350	294.4
6		12352	1265.4
7		12353	89.0
8		12354	1079.4
9		12355	459.4
10		12356	2487.4

- 고객별 평균 거래 금액 계산
 - 고객별 평균 거래 금액을 구하기 위해 1) data 테이블을 user_rf 테이블과 조인(LEFT JOIN) 한 후, 2) purchase_cnt 로 나누어서 3) user_rfm 테이블로 저장하기

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_rfm` AS

SELECT

rf.CustomerID AS CustomerID,

rf.purchase_cnt,

rf.item_cnt,

rf.recency,

ut.user_total,

ROUND(ut.user_total / rf.purchase_cnt, 1) AS user_average

FROM `proven-hash-466600-d5.modulabs_project.user_rf` rf

LEFT JOIN (SELECT

CustomerID,

ROUND(SUM(Quantity * UnitPrice), 1) AS user_total

FROM `proven-hash-466600-d5.modulabs_project.data`

WHERE CustomerID IS NOT NULL

GROUP BY CustomerID) ut

ON rf.CustomerID = ut.CustomerID;
```

[결과]

이 문으로 이름이 user_rfm인 새 테이블이 생성되었습니다.

행 //	CustomerID ▼//	purchase_cnt 🔻	item_cnt ▼ //	recency ▼ //	user_total ▼ //	user_average ▼//
1	12713	1	505	0	794.5	794.5
2	15520	1	314	1	343.5	343.5
3	13436	1	76	1	196.9	196.9
4	13298	1	96	1	360.0	360.0
5	14569	1	79	1	227.4	227.4
6	17436	1	12	1	17.4	17.4
7	14204	1	72	2	150.6	150.6
8	15195	1	1404	2	3861.0	3861.0
9	15471	1	249	2	442.1	442.1
10	12442	1	181	3	144.1	144.1

RFM 통합 테이블 출력하기

• 최종 user_rfm 테이블을 출력하기

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.user_rfm`
```

[결과]

행	1	CustomerID ▼	purchase_cnt 🔻	item_cnt ▼	recency ▼	user_total ▼ //	user_average ▼//
	1	12713	1	505	0	794.5	794.5
	2	15520	1	314	1	343.5	343.5
	3	13436	1	76	1	196.9	196.9
	4	13298	1	96	1	360.0	360.0
	5	14569	1	79	1	227.4	227.4
	6	17436	1	12	1	17.4	17.4
	7	14204	1	72	2	150.6	150.6
	8	15195	1	1404	2	3861.0	3861.0
	9	15471	1	249	2	442.1	442.1
	10	12442	1	181	3	144.1	144.1

11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

• 1) 고객 별로 구매한 상품들의 고유한 수를 계산하기

2)

user_rfm 테이블과 결과를 합치기

3)

user_data 라는 이름의 테이블에 저장하기

```
CREATE OR REPLACE TABLE `proven-hash-466600-d5.modulabs_project.user_data` AS

WITH unique_products AS (

SELECT

CustomerID,

COUNT(DISTINCT StockCode) AS unique_products

FROM `proven-hash-466600-d5.modulabs_project.data`

WHERE CustomerID IS NOT NULL

GROUP BY CustomerID
)

SELECT ur.*, up.* EXCEPT (CustomerID)

FROM `proven-hash-466600-d5.modulabs_project.user_rfm` AS ur

JOIN unique_products AS up
```

```
ON ur.CustomerID = up.CustomerID;

-- 검증
WITH unique_products AS (
SELECT
CustomerID,
COUNT(DISTINCT StockCode) AS unique_products
FROM `proven-hash-466600-d5.modulabs_project.data`
GROUP BY CustomerID
)
SELECT * from
unique_products
LIMIT 10;
```

[결과]



2. 평균 구매 주기

- 고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)
 - 。 균 구매 소요 일수를 계산하고, 그 결과를 user_data 에 통합

```
CREATE OR REPLACE TABLE 'proven-hash-466600-d5.modulabs_project.user_data' AS
WITH purchase_intervals AS (
SELECT
 CustomerID,
 IFNULL(ROUND(AVG(interval_), 2), 0) AS average_interval
FROM (
 SELECT
   CustomerID,
  DATE_DIFF(DATE(InvoiceDate), LAG(DATE(InvoiceDate)) OVER (PARTITION BY CustomerID ORDER BY DATE(InvoiceDate)), DAY
   `proven-hash-466600-d5.modulabs_project.data`
  WHERE CustomerID IS NOT NULL
GROUP BY CustomerID
SELECT u.*, pi.* EXCEPT (CustomerID)
FROM `proven-hash-466600-d5.modulabs_project.user_data` AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

[결과]

① 이 문으로 이름이 user_data인 테이블이 교체되었습니다.

행 //	CustomerID 🕶	purchase_cnt /	item_cnt 🕶 //	recency •	user_total ▼ //	user_average 🔨	unique_products	average_interval
1	16344	1	18	158	101.0	101.0	1	0.0
2	13270	1	200	366	590.0	590.0	1	0.0
3	16257	1	1	176	22.0	22.0	1	0.0
4	15668	1	72	217	76.0	76.0	1	0.0
5	15510	1	2	330	250.0	250.0	1	0.0
6	16323	1	50	196	208.0	208.0	1	0.0
7	16738	1	3	297	4.0	4.0	1	0.0
8	14576	1	12	372	35.0	35.0	1	0.0
9	14090	1	72	324	76.0	76.0	1	0.0
10	16990	1	100	218	179.0	179.0	1	0.0

3. 구매 취소 경향성

• 고객의 취소 패턴 파악하기

1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수

2) 취소 비율(cancel_rate): 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율

 취소 빈도와 취소 비율을 계산하고 그 결과를 user_data 에 통합하기 (취소 비율은 소수점 두번째 자리)

```
CREATE OR REPLACE TABLE 'proven-hash-466600-d5.modulabs_project.user_data' AS
WITH TransactionInfo AS (
SELECT.
 CustomerID.
 COUNT(DISTINCT InvoiceNo) AS total_transactions,
 COUNT(DISTINCT IF(STARTS_WITH(InvoiceNo, 'C'), InvoiceNo, NULL)) AS cancel_frequency
FROM `proven-hash-466600-d5.modulabs_project.data`
WHERE CustomerID IS NOT NULL
GROUP BY CustomerID
SELECT
u.*,
t.* EXCEPT(CustomerID),
ROUND(IFNULL(t.cancel_frequency / t.total_transactions, 0), 2) AS cancel_rate
FROM `proven-hash-466600-d5.modulabs_project.user_data` AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```

[결과]



• 다양한 컬럼들을 활용하여 고객의 구매 패턴과 선호도를 보다 심층적으로 이해할 수 있도록 최종적으로 user_data 를 출력하기

```
SELECT *
FROM `proven-hash-466600-d5.modulabs_project.user_data`
ORDER BY user_total DESC;
```

[결과]

행 //	CustomerID	▼ purchase //	item_cnt ▼ //	recency 🕶	user_total ▼	user_average	unique_p	average_i //	total_tran	cancel_fr //	cancel_rate /
1	14646	73	196556	1	278778.0	3819.0	699	0.16	73	1	0.01
2	18102	60	64124	0	259657.0	4328.0	150	0.82	60	0	0.0
3	17450	49	69021	8	189576.0	3869.0	124	1.02	49	3	0.06
4	14911	242	76823	1	128768.0	532.0	1791	0.05	242	44	0.18
5	12415	24	76946	24	123638.0	5152.0	443	0.39	24	4	0.17
6	14156	64	56896	9	113686.0	1776.0	714	0.24	64	10	0.16
7	17511	45	63014	2	88138.0	1959.0	465	0.33	45	14	0.31
8	16684	30	49391	4	65920.0	2197.0	119	1.24	30	2	0.07
9	13694	57	61904	3	62962.0	1105.0	367	0.6	57	7	0.12
10	16029	66	33687	38	60370.0	915.0	43	1.22	66	4	0.06

회고

Keep: BigQuery를 활용해 대용량 데이터를 효율적으로 가공하고, 단계별 쿼리를 수행해 최종적으로 하나의 user_data 테이블에 통합해 분석용 데이터넷을 완성했다.

Problem : 데이터 삭제과정에서 실수가 있어 다시 데이터를 불러와 진행, 컬럼 중복 문제(average_interval) 에러가 발생할때마다 불필요하게 시간이 소요되어 아직 부족한점을 많이 느꼈다. 그래서 쿼리가 정상 실행되어도 계속 결과를 확인하고 의심하는 시간이 많았다.

Try : Recency, Frequency, Monetary + 부가 지표까지 한 번에 생성하는 SQL 쿼리도 시도하면 좋을 것 같고 시각화, 머신러닝 적용 등 내일 진행할 내용까지 포함하지 않아 아직 마무리가 덜 된 느낌... 무엇보다 아직 쿼리 작성에 익숙하지 않다는 생각이 계속 들어서 연습이 더 필요할 것 같다.